

Lecture 22: Time-stepping schemes I.

- For the next two lectures we will consider ODEs of the form

$$\frac{\partial u}{\partial t} = L \left(t, u, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2} \right)$$

- Each partial differential equation needs to be considered and classified individually with regards to stability. For this lecture we will thus focus on the one-way wave equation in 1D.
- **The method of lines** is a class of methods which use the data at a single slice of time to generate a solution Δt in the future, which in the next iteration is used to obtain the solution slice at $2\Delta t$.

-open file Fig1Lec22.pdf

Today we will focus on the one-way wave equation

$$\frac{\partial u}{\partial t} = c \frac{\partial u}{\partial x}$$

Applying the **Method of Lines (MOL) with a Forward Euler time-stepping scheme** (using first order forward difference for the derivative in time and second order central difference in space), denoting $u_n^{(m)} = u(t_m, x_n)$

-open Fig2Lec22.pdf

$$\begin{aligned} u_n^{(m+1)} &= u_n^{(m)} + \frac{c\Delta t}{2\Delta x} \left(u_{n+1}^{(m)} - u_{n-1}^{(m)} \right) \\ &= u_n^{(m)} + \frac{\lambda}{2} \left(u_{n+1}^{(m)} - u_{n-1}^{(m)} \right) \end{aligned}$$

Courant, Friedrichs, and Levy (CFL) condition

$$\rightarrow \lambda = \frac{c\Delta t}{\Delta x}$$

- The CFL condition is essential to controlling the accuracy and stability of the method.
- Decreasing Δx keeping Δt fixed results in an increasing λ which can lead to instabilities if λ is too large.
- Decreasing Δt keeping Δx fixed results in a decreasing λ which can be stable, however it could be so small that it is not efficient.
- The key to accurate, efficient and stable methods is in the CFL number. Ultimately you want to choose Δt and Δx large so you can compute quickly and efficiently without instabilities.

The Forward Euler time-stepping scheme for the one-way wave equation will show instabilities eventually for any choice of λ .

Using second order central differencing in time as well as space results in a well known method called **leap-frog (2,2)** since it is $O(\Delta t^2)$ accurate in time and $O(\Delta x^2)$ in space.

-open file Fig3Lec22.pdf

$$\begin{aligned}u_n^{(m+1)} &= u_n^{(m-1)} + \frac{c\Delta t}{\Delta x} \left(u_{n+1}^{(m)} - u_{n-1}^{(m)} \right) \\ &= u_n^{(m-1)} + \lambda \left(u_{n+1}^{(m)} - u_{n-1}^{(m)} \right)\end{aligned}$$

The leap-frog(2,2) method applied to the one-way wave equation is stable for $\lambda \leq 1$

Take the same second order central differencing in time and fourth order central differencing in space (improves accuracy in space) is called leap-frog (2,4)

$$\begin{aligned}
 & u_n^{(m+1)} - u_n^{(m-1)} \\
 &= \lambda \left[\frac{4}{3} \left(u_{n+1}^{(m)} - u_{n-1}^{(m)} \right) - \frac{1}{6} \left(u_{n+2}^{(m)} - u_{n-2}^{(m)} \right) \right]
 \end{aligned}$$

The leap-frog(2,4) method applied to the one-way wave equation is stable for $\lambda \leq 0.707$

Implicit scheme: Backward Euler method,
is stable for all λ !

$$\begin{aligned}
 u_n^{(m+1)} &= u_n^{(m)} + \frac{c\Delta t}{2\Delta x} \left(u_{n+1}^{(m+1)} - u_{n-1}^{(m+1)} \right) \\
 &= u_n^{(m)} + \frac{\lambda}{2} \left(u_{n+1}^{(m+1)} - u_{n-1}^{(m+1)} \right)
 \end{aligned}$$

Rearranging so that all $(\cdot)^{(m+1)}$ terms are on one side and all $(\cdot)^{(m)}$ terms on the other

$$\frac{\lambda}{2}u_{n-1}^{(m+1)} + u_n^{(m+1)} - \frac{\lambda}{2}u_{n+1}^{(m+1)} = u_n^{(m)}$$

Which can be written as a matrix system (with a tridiagonal matrix \mathbf{A})

$$\mathbf{A}\mathbf{u}^{(m+1)} = \mathbf{u}^{(m)}$$

Two other methods of interest not covered today but you should read

- Lax-Wendroff
- Predictor Corrector method: MacCormack scheme

In MATLAB (Forward Euler and Leap-Frog(2,2)) solve the one-way wave equation with $c = 1$, for $x \in [-L/2, L/2]$ and periodic BCs.

```
% initialize grid size, CFL number and time
L=20;
n=200;
x2=linspace(-L/2,L/2,n+1);  x=x2(1:n);
dx=x(2)-x(1);

dt=0.2;
CFL=dt/dx

Time=4;
time_steps=Time/dt;
t=0:dt:Time;

% initial conditions (gaussian)
u0=exp(-x.^2)';
usol(:,1)=u0;
u1=exp(-(x+dt).^2)';
usol(:,2)=u1;
```

```

% sparse matrix for derivative term
e1=ones(n,1);
A=spdiags([-e1 e1],[-1 1],n,n);

% periodic boundary conditions
A(1,n)=-1; A(n,1)=1;

% leap frog (2,2) or euler iteration scheme
for j=1:time_steps-1

% forward euler (start at u1)
u2 = u1 + 0.5*CFL*A*u1;
u1 = u2;

% leap-frog (2,2) (needs u0 and u1)
% u2 = u0 + CFL*A*u1; % leap frog (2,2)
% u0 = u1; u1 = u2; % leap frog (2,2)

% save solution at time slice in matrix
usol(:,j+2)=u2;
end

```

```
% plot the data
waterfall(x,t,usol');
map=[0 0 0];
colormap(map);

% set x and y limits and fontsize
set(gca,'Xlim',[-L/2 L/2],'Xtick',[-L/2 0 L/2], ...
'FontSize',[20]);
set(gca,'Ylim',[0 Time],'Ytick',[0 Time/2 Time], ..
'FontSize',[20]);
view(25,40)

% set axis labels and fonts
xl=xlabel('x');
yl=ylabel('t');
zl=zlabel('u');
set(xl,FontSize,[20]);
set(yl,FontSize,[20]);
set(zl,FontSize,[20]);

% print jpeg at screen resolution
print -djpeg -r0 fig.jpg
```